Standard Library Improvements

Chapel Team, Cray Inc.

Chapel version 1.13

April 7, 2016

Safe Harbor Statement

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts. These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.

Outline

- Standard Library vs. Package Modules
- Reflection Library Module
  - Querying Function Resolution
  - Inspecting Fields
- Improvements to the Random Standard Module
- JAMA: Linear Algebra module
- FileSystem/Path Module Improvements
- I/O Improvements for JSON
- Other I/O Improvements
- Spawn Module Improvements
- Other Library Improvements

Standard Library vs. Package Modules

Libraries vs. Packages: Background & This Effort

Background:

- Most user-facing libraries we've distributed have lived in one directory:

  `$CHPL_HOME/modules/standard/`

This Effort:

- Over time, it's seemed that we have two classes of standard modules:
  - standard library modules
    - those that any Chapel implementation ought to support
    - e.g., 'Math', 'FileSystem', 'Reflection', …
  - package modules
    - those representing community codes or optional features
      - e.g., 'FFTW', 'LAPACK', 'HDFS', …
      - ultimately these should be managed by a distributed package manager
    - we've also moved some immature libraries here for the time being
      - e.g., 'Sort', 'Search', 'Norm', …

Libraries vs. Packages: Status and Next Steps

Status:

- Split standard modules into two directories, along with their docs:

  libraries: `$CHPL_HOME/modules/standard`

  - see http://chapel.cray.com/docs/latest/modules/modules.html

  packages: `$CHPL_HOME/modules/packages`

  - see http://chapel.cray.com/docs/latest/modules/packages.html
- Compiler populates module search path with both directories

Next Steps:

- Improve immature packages and promote to standard library
- Create a distributed package manager (see “Ongoing Efforts” slides)

Reflection Library Module

Reflection: Background and This Effort

Background:

- Chapel already supports generic programming:
  - where clauses, type functions, param functions, and param for-loops
- Yet, it can also be useful to make explicit queries
  - about procedures or methods that are available
  - about fields in records or classes
- Such introspection has proven valuable in C++
  - SFINAE and enable-if in C++
- Chapel's internal modules already make such queries
  - but not in ways that were intended for end-users

This Effort:

- Provide user-facing introspection in a new 'Reflection' module
  - inspecting which procedures or methods are available
  - inspecting fields in records or classes

Reflection: Querying Function Resolution

Can Resolve: Motivation and This Effort

Motivation:

- Suppose we're writing a sorting library
  - Uses a *compare* routine, where *compare*(a,b) returns an int that's…

    < 0 if a < b

    == 0 if a == b

    > 0 if a > b
- For usability reasons, we would like the library to
  - optionally allow *compare* to be written as a method
  - fall back on calls to < if no *compare* method/function is provided

This Effort:

- Adds *canResolve* and *canResolveMethod* functions

Can Resolve: Library Example

```
use Reflection;

private proc do_compare(a, b) {
  if canResolveMethod(a, "compare", b) {
    // use a.compare(b) if possible
    return a.compare(b);
  } else if canResolve("compare", a, b) {
    // if not, use compare(a,b)
    return compare(a, b);
  } else {
    //  otherwise, fall back on a version using <
    if a < b then
      return -1;
    else if b < a then
      return 1;
    else return 0;
  }
}
```

Can Resolve: Usage Example

```
record MyRecord {
  var x:int;
}

// Client of sorting library could provide
// any or all of these comparison routines.
proc MyRecord.compare(b: MyRecord) {
  return this.x - b.x;
}
proc compare(a: MyRecord, b: MyRecord) {
  return a.x - b.x;
}
proc <(a: MyRecord, b: MyRecord) {
  return a.x < b.x;
}
```

Can Resolve: Impact, Status, and Next Steps

Impact:

- Enables generic libraries to respond to what functions are available

Status:

- Implemented, documented, fully functional

Next Steps:

- Replace current internal uses of ‘tryToken’ with ‘canResolve’
  - this was the previous (and weaker) internal means of making such choices
- Get user feedback on ‘canResolve’

Reflection: Inspecting Fields

Inspecting Fields: Motivation

Motivation:

- Suppose that we're writing an I/O library with a particular format, e.g.
  - JSON
  - XML
  - YAML
  - BSON
  - MessagePack
  - Protocol Buffers
  - Apache Thrift
- For usability reasons, want the library to work with any record or class

This Effort:

- Adds field introspection routines
  - *numFields, getFieldName*, *getField*, *getFieldIndex*, *hasField*

Inspecting Fields: Library Example

```
use Reflection; use Types;

private proc do_outputYaml(obj:?t, indent:string) {
  for param i in 1..numFields(t) {      // param loop over fields
    var field = getField(obj, i);       // query value of ith  field
    var fieldName = getFieldName(t, i);  // query name of ith field
    if isString(field) || isNumeric(field) {
      // Output numbers and strings directly
      writeln(indent, fieldName, ": ", field);
    } else if isRecord(field) || isClass(field) {
      // Recurse to output a record or class.
      writeln(indent, fieldName, ":");
      do_outputYaml (field, indent + "    ");
    } else compilerError("unhandled type ", field.type:string);
  }
}
proc outputYaml(obj) {
  writeln("---"); do_outputYaml(obj, ""); writeln("...");
}
```

Inspecting Fields: Usage Example

```
var r = new Invoice(number=22, total=135.17,
                    new Customer("Dorothy", "Gale"));
outputYaml(r);

record Customer {
  var first_name:string;
  var family_name:string;
}
record Invoice {
  var number:int;
  var customer:Customer;
  var total:real;
}
```

```
---
number: 22
customer:
    first_name: Dorothy
    family_name: Gale

total: 135.17
...
```

Inspecting Fields: Impact, Status, and Next Steps

Impact:

- Enables generic libraries to adapt to fields in records and classes

Status:

- Implemented, documented
- *getField* does not currently return a mutable value
  - primitive can be used for now as a workaround
  - *getFieldRef* implemented but didn't make it into v1.13 (on master now)

Next Steps:

- replace uses of primitives in ChapelIO with Reflection module routines
- consider changing these functions to methods / type methods

Improvements to the Random Standard Module

Random: Background

- **<= 1.12 only had NPB Random Number Generator (RNG)**
- **NPB RNG has issues**
  - it's a 46-bit Linear Congruential Generator (LCG)
  - it only supports generating real(64), imag(64), complex(128)
  - it's implemented in terms of double-precision arithmetic
    - i.e., has lower than ideal performance if converting reals to integers
  - it's not suitable for Monte Carlo*
  - it fails 41/144 statistical tests in TestU01's Crush suite
- **Yet, the NPB RNG supports *jump-ahead***
  - supports deterministic parallel RNG use cases
- **Have long intended to address these issues**
  - Users have requested better Random support as well

* = see Click, Kaminski, and Liu, "Quality of random number generators significantly affects results of Monte Carlo simulations for organic and biological systems"

Random: Some other RNGs

- **Mersenne Twister**
  - popular
  - used in Python
  - we didn't find a version that supported jump-ahead that we liked
    - came with disclaimers
    - raised licensing challenges (GPL, vague/missing licenses)

- **PCG: Permuted Congruential Generator**
  - reasonably new
  - supports jump-ahead
  - simple implementation
  - C implementation available with permissive license (Apache)

Random: This Effort

- **Implemented PCG RNG purely in Chapel code**
  - it was simple enough not to require wrapping the C library
  - main PCG RNG has 64 bits of state and generates 32 bits at a time
  - generate >32 bits by splicing RNGs with different increments
  - Adds functionality to the C PCG implementation
    - generates real values (via multiplying by 2**-64)
    - generates bounded integer values in way that supports jump-ahead
- **Refactored Random standard module**
  - Parallel *RandomStream* available with either NPB or PCG RNG
  - PCG is now the default
  - Low-level PCG module is also available, providing interface like C's
- **Added new functionality**
  - *shuffle*, *permute*, generate random integer
- **Documented *RandomStream.iterate***

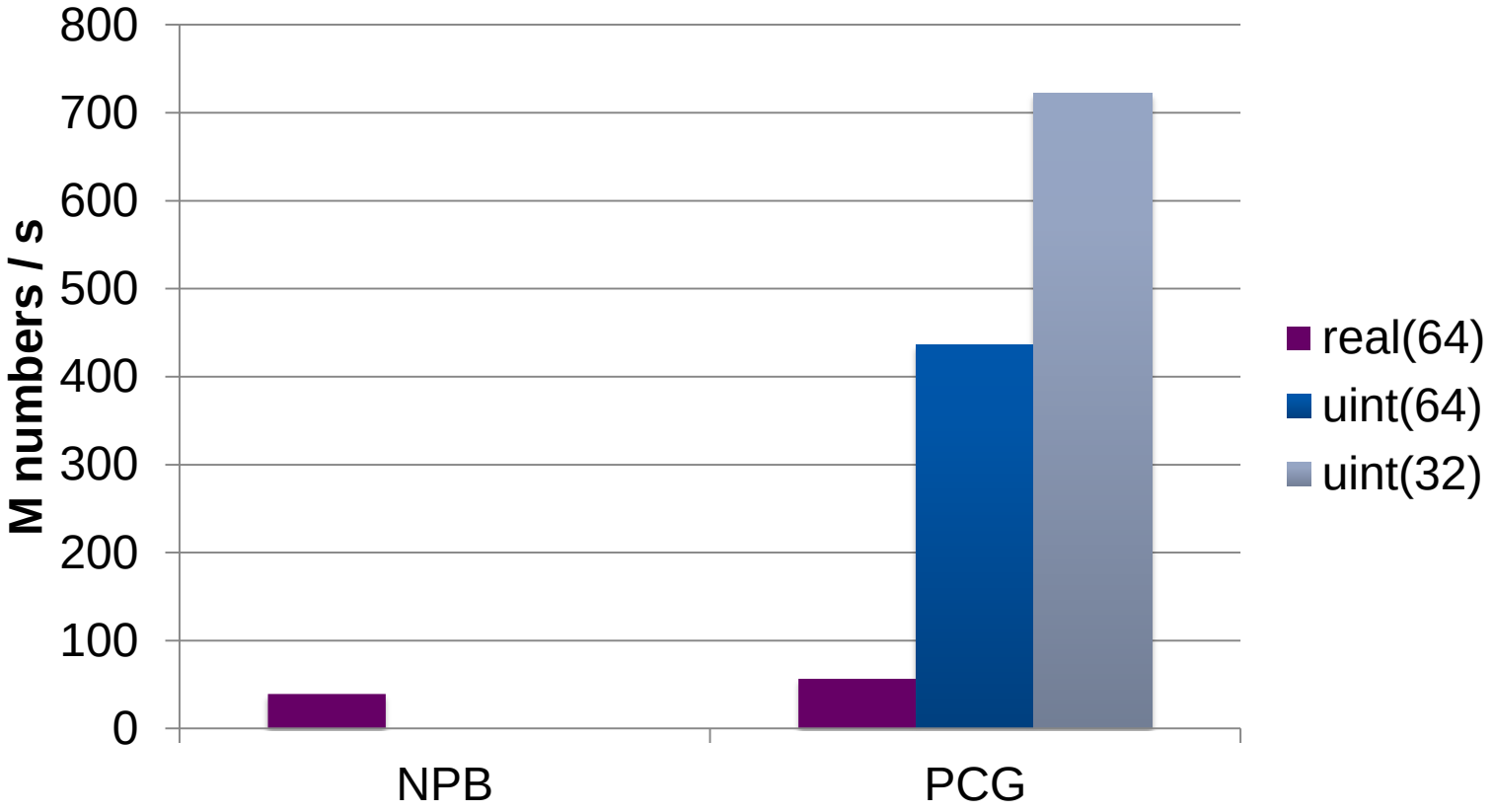
Random: Impact

- **Tested statistical properties with TestU01's Crush suite:**
  - NPB failed 41/144 tests for real(64)
  - PCG passed all tests for real(64), uint(64)
  - PCG failed 1 test for real(32)
    - as does the C PCG implementation with the same seed

⇒ PCG offers much improved statistical properties

Random: Impact

- **PCG is faster and more flexible than NPB**
  - 1.4x when generating doubles; able to generate integers directly



Random: Impact

Status:

- PCG RNG is the new default in 1.13
- Better performance and statistical properties
- Integer output supported

Next Steps:

- Improve API based on review, user feedback, Chapel improvements
  - e.g., leader-follower improvements required to zipper RNGs naturally
- Support additional RNGs as necessary / desired

JAMA: Linear Algebra module
(contributed by Chris Taylor)

JAMA: Background and This Effort

Background:

- JAMA: Linear Algebra package originally for Java
  - Aimed at creation/manipulation of real, dense matrices
  - Developed by MathWorks and NIST
  - Ported to Chapel
- Focuses on 5 fundamental matrix decompositions:
  - Cholesky – for symmetric, positive definite matrices
  - LU – for rectangular matrices
  - QR – for rectangular matrices
  - Eigenvalue – for symmetric and nonsymmetric square matrices
  - Singular – for rectangular matrices

This Effort:

- Ported JAMA to Chapel

JAMA: LU Decomposition Example

- **Example usage:**

```
use LinearAlgebraJama;

var A = …; // Initialize A.  We'll compute L and U from A such that L*U = A
var LU = A.lu();
var L = LU.getL();
var U = LU.getU();
… // Using L and U, solve Ax = b for x given b (for instance)
```

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}$$

JAMA: Eigenvalue Example

- **Example 2:**

```
use LinearAlgebraJama;

var A = …; // Initialize A. If symmetric, V*D*V' = A
var Eig = A.eig();
var D = Eig.getD(); // an eigenvalue matrix
var V = Eig.getV(); // an eigenvector matrix
… // Using D and V, find the inverse of A (for instance)
```

JAMA: Next Steps

- **Create online documentation**
  - Current document comments are javadoc style rather than chpldoc
- **Connect to LAPACK and BLAS modules**
  - Once BLAS module routines finished
- **Design/implement sparse matrix solution**
- **Use as starting point for computer algebra module**
  - Similar to Python's Theano, NumPy
  - Aimed at dense vector/matrix computations
  - Also would make use of LAPACK/BLAS routines

FileSystem/Path Module Improvements

FileSystem/Path: Background and This Effort

Background:

- Recent releases added *FileSystem* and *Path* modules
  - *FileSystem* was nearly complete, but missing a few routines
  - *Path* was mostly empty, waiting on strings to mature

This Effort:

- Complete *FileSystem*
- Continue with *Path* now that strings are in good shape

FileSystem/Path: Status and Next Steps

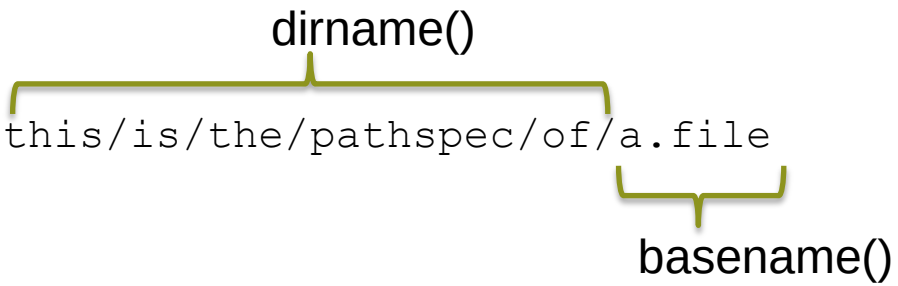
Status:

FileSystem: Finished remaining routines

- **rmTree()**: remove directory and contents
- **moveDir()**: move directory and contents to new location

Path: Added a few more routines

- **basename()**: last component of given path
- **dirname()**: all but last component of given path
- **splitPath()**: divide path into (dirname, basename)
- 10 functions remain unimplemented
  - **absPath()**, **joinPath()**, etc.



Next Steps: complete remaining *Path* routines

Spawn Module Improvements

Spawn Module Improvements

Background: Added 'Spawn' standard module in v1.12

- had a bug where *communicate()* did not function correctly
- had several bugs related to interrupted system calls
- did not include functions for sending signals to subprocesses

This Effort: Improved the Spawn module

- fixed *communicate()*
- made *wait()* optionally call it when *buffer*=true via new argument:

```
proc subprocess.wait(out error: syserr, buffer=true) …
```

- fixed problems with interrupted system calls
- added support for sending signals (contributed by Nick Park)

Impact: Spawn module is much more stable and usable

Next Steps: Write more programs using the Spawn module!

I/O Improvements for JSON

JSON Improvements: Background & This Effort

Background:

- I/O module supports *writeln*, *writef*, …
- during v1.12, there was an ongoing effort to improve JSON I/O

This Effort:

- Improved JSON support with ~ in format strings to skip fields
  - see example on following slides

Example Tweet in JSON format

- **Tweets have 34 top-level fields**
  - including nested structures containing additional fields

{ "coordinates": null, "created_at": "Fri Oct 16 16:00:00 +0000 2015", "favorited": false, "truncated": false, "id_str": "28031452151", **"entities": {** "urls": [{ "expanded_url": null, "url": "http://chapel.cray.com", "indices": [69, 100] }], "hashtags": [ ], **"user_mentions": [{** "name": "Cray Inc.", "id_str": "23424245", **"id": 23424245**, "indices": [25, 30], "screen_name": "cray" **}] },** "in_reply_to_user_id_str": null, "text": "Let's mention the user @cray -- here is an embedded url ..... http://chapel.cray.com", "contributors": null, **"id": 28039652140**, "retweet_count": null, "in_reply_to_status_id_str": null, "geo": null, "retweeted": false, "in_reply_to_user_id": null, **"user": {** "profile_sidebar_border_color": "C0DEED", "name": "Cray Inc.", "profile_sidebar_fill_color": "DDEEF6", "profile_background_tile": false, "profile_image_url": "http://a3.twimg.com/profile_images/2342452/icon_normal.png", "location": "Seattle, WA", "created_at": "Fri Oct 10 23:10:00 +0000 2008", "id_str": "23502385", "follow_request_sent": false, "profile_link_color": "0084B4", "favourites_count": 1, "url": "http://cray.com", "contributors_enabled": false, "utc_offset": -25200, **"id": 23548250**, "profile_use_background_image": true, "listed_count": 23, "protected": false, "lang": "en", "profile_text_color": "333333", "followers_count": 1000, "time_zone": "Mountain Time (US & Canada)", "verified": false, "geo_enabled": true, "profile_background_color": "C0DEED", "notifications": false, "description": "Cray Inc", "friends_count": 71, "profile_background_image_url": "http://s.twimg.com/a/2349257201/images/themes/theme1/bg.png", "statuses_count": 302, "screen_name": "gnip", "following": false, "show_all_inline_media": false **},** "in_reply_to_screen_name": null, "source": "web", "place": null, "in_reply_to_status_id": null **}**

JSON Improvements: Reading Tweet subsets

```
// define Chapel records whose fields reflect only
// the portions of the JSON data we care about

record TweetUser {
  var id: int;
}
record TweetEntities {
  var user_mentions: list(TweetUser);
}
record User {
  var id: int;
}
record Tweet {
  var id: int,
      user: User,
      entities: TweetEntities;
}
```

```
proc process_json(…) {
  var tweet: Tweet;

  while true {
    // "%~jt" format string:
    //    j: JSON format
    //    t: any record
    //    ~: skip other fields
    got = logfile.readf("%~jt",
                        tweet,
                        error=err);
    if got && !err then
      handle_tweet(tweet);
    if err == EFORMAT then ...;
    if err == EEOF then break;
  }
```

JSON Improvements: Impact and Next Steps

Impact:

- JSON support is improved in version 1.13

Next Steps:

- Support formatting extensions to *channel*
  - since not all formats will be supported directly as JSON is

Other I/O Improvements

Other I/O Improvements

- **HDFS package now supports libhdfs3**
  (contributed by Chris Taylor)
- **Removed *Reader* and *Writer* types in favor of *channels***
- **Made *readbits*/*writebits* accept any integral argument**
- **List module now supports JSON format**
- **Default I/O routines now ignore 'param' fields**
- **channels now support an isClosed() method**

Other Library Improvements

Other Library Improvements

- **Time.sleep() now optionally supports a unit argument**
  (contributed by Nick Park)
- **exit() can now be called without arguments**
  (contributed by Kushal Singh)
- **Also, aforementioned library routines for strings:**
  http://chapel.cray.com/docs/1.13/modules/internal/String.html

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, and URIKA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

CRAY

THE SUPERCOMPUTER COMPANY